Heterogeneous Anomaly Detection for Software Systems via Semi-supervised Cross-modal Attention

Cheryl Lee*, Tianyi Yang*, Zhuangbin Chen†, Yuxin Su†, Yongqiang Yang‡, and Michael R. Lyu*
*Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China.
Email: cheryllee@link.cuhk.edu.hk, {tyyang, lyu, zbchen}@cse.cuhk.edu.hk
†Sun Yat-sen University, Guangzhou, China. Email: suyx35@mail.sysu.edu.cn
‡Computing and Networking Innovation Lab, Cloud BU, Huawei. Email: yangyongqiang@huawei.com

***Abstract*—Prompt and accurate detection of system anomalies is essential to ensure the reliability of software systems. Unlike manual efforts that exploit all available run-time information, existing approaches usually leverage only a single type of monitoring data (often logs or metrics) or fail to make effective use of the joint information among different types of data. Consequently, many false predictions occur. To better understand the manifestations of system anomalies, we conduct a systematical study on a large amount of heterogeneous data, i.e., logs and metrics. Our study demonstrates that logs and metrics can manifest system anomalies collaboratively and complementarily, and neither of them only is sufficient. Thus, integrating heterogeneous data can help recover the complete picture of a system's health status. In this context, we propose Hades, the first end-to-end semi-supervised approach to effectively identify system anomalies based on heterogeneous data. Our approach employs a hierarchical architecture to learn a global representation of the system status by fusing log semantics and metric patterns. It captures discriminative features and meaningful interactions from heterogeneous data via a cross-modal attention module, trained in a semi-supervised manner. We evaluate Hades extensively on large-scale simulated data and datasets from Huawei Cloud. The experimental results present the effectiveness of our model in detecting system anomalies. We also release the code and the annotated dataset for replication and future research.**

***Index Terms*—Software System, Anomaly Detection, Cross-modal Learning**

I. Introduction

Recent years have witnessed the scale and complexity of software systems expand dramatically. However, anomalies are inevitable in large-scale software systems, resulting in considerable revenue and reputation loss [1]. The core competence of service providers stems from guaranteeing the reliability of software systems, where automated anomaly detection is a primary step and has been picked up extensively within the community. In real-world software systems, many types of monitoring data, including metrics, logs, alerts, and traces, play an essential role in software reliability engineering [2], [3]. In particular, metrics and logs have been widely used for anomaly detection. Metrics (e.g., response time, number of threads, CPU usage) are real-valued time series measuring the system status. Logs are semi-structured text messages printed by logging statements to record the system's run-time status.

Tremendous efforts have been devoted to detecting anomalies automatically since manual troubleshooting is impractical and error-prone. Some approaches rely on metrics [4]–[6], while others rely on logs [7]–[9]. However, as a single source of information is often insufficient to depict the status of a software system precisely, existing methods could produce many false predictions [10] The popularity of large-scale distributed systems worsened the situation, where anomaly patterns are more complex. Intuitively, combining different sources of monitoring data can allow fuller utilization of run-time information to analyze the system status holistically.

To verify our intuition, we study the characteristics of system anomalies incurred by typical faults based on a large amount of heterogeneous monitoring data. The data are generated via fault injection on Apache Spark, where we run various workloads and inject 21 typical types of faults. Compared to existing open-access datasets, e.g., [10]–[13], ours possesses temporally aligned heterogeneous run-time information with rich semantics and annotations (i.e., abnormal or not).

Based on the study, we obtain three interesting findings:

- Though the presence of critical logs often indicates problems, their absence does not necessarily imply a healthy system status. An important reason is that sometimes determining where and how to place an informative log statement is difficult [14].
- In some cases, faults do not affect metrics, while in other cases, metrics exhibit unusual patterns (e.g., jitters) even if the system is experiencing minor performance fluctuations instead of faults. Hence, simply identifying anomalous metric patterns is insufficient.
- Faults can cause unexpected behaviors involving either logs or metrics, or both of them. So the two data sources should be analyzed comprehensively to reveal the actual anomalies.

These findings necessitate considering heterogeneous data for anomaly detection, and high-level patterns (i.e., log semantics and metric patterns) of the observed data deserve full attention.

However, we identify three challenges in extracting and integrating essential information from heterogeneous data. *(1) Complex intra-modal information.* Logs reflect system anomalies mainly through their semantics (e.g., keywords) and sequential dependencies across events. Besides, metrics reflect diverse aspects of the system status, and metrics of different aspects tend to develop distinct behavior patterns. For example, when a system works normally, the disk usage often moves steadily, while the CPU usage can fluctuate

Yuxin Su is the corresponding author.

1558-1225/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSE48619.2023.00148

dramatically. Such complex and diverse data patterns call for a model to be highly competent in information processing and feature extraction. *(2) Significant inter-modal gap.* Logs and metrics are in different forms, i.e., textual and time series. Such a discrepancy poses a huge challenge to effectively using the joint information for downstream anomaly detection. To this end, it is critical to align the log semantics and metric patterns. *(3) Trade-off between cost and accuracy.* Supervised approaches are effective but require high-quality labels. Annotating massive logs and metrics is prohibitively difficult, costly, and time-consuming, so the requirement of annotations is usually the bottleneck of putting supervised approaches into practice. Though unsupervised learning avoids labeling by mining inherent trends of unlabeled data to discover anomalies, it suffers inaccuracies with less human oversight and ignorance of valuable domain expertise.

To tackle these challenges, we propose **Hades**, a Heterogeneous Anomaly DEtector via Semi-supervised learning for large-scale software systems, equipped with a novel cross-modal attention mechanism. The key idea is to learn a discriminative representation of the system status based on logs and metrics with limited labeled data for training. Hades first captures intra-modal dependencies using a hierarchical architecture. Then it generates a global representation of the most discriminative latent information of logs and metrics via a modal-wise attentive fusion module.

Specifically, Hades involves four components for data modeling and engages semi-supervised training. The components are: (1) For logs, we adopt the FastText algorithm [15] and Transformer [16] to model lexical semantics and sequential dependencies of logs. (2) We employ a hierarchical encoder to learn metric representations based on the causal convolution network [17]. It jointly learns aspect-oriented temporal dependencies, cross-metric relationships, and inter-aspect correlations. (3) We design a novel modal-wise attention mechanism to facilitate learning meaningful intra- and inter-modal properties. (4) Finally, the framework infers the system status and triggers an alarm upon detecting anomalies. The semi-supervised training comprises two phases: First, we apply a few labeled data to train the initial model and then pseudo-label the remaining unlabeled data via the current training model. Second, the model is updated using both labeled and pseudo-labeled data with high confidence until convergence.

We evaluate Hades using one simulated and two datasets from Huawei Cloud. The experimental results demonstrate the superiority of Hades, which achieves an average F1-score of 0.933 and outperforms all state-of-the-art competitors, including log-based and metric-based ones. Extensive ablation experiments further confirm the effectiveness of our designs (i.e., exploiting heterogeneous information, cross-modal learning, attentive fusion, and intra-modal feature extraction).

In summary, the main contributions of this paper are:

- We systematically study how heterogeneous data manifest system anomalies. To our best knowledge, we are the first to point out the collaborative and complementary relationship of logs and metrics in manifesting anomalies.
- We propose the first end-to-end semi-supervised approach, Hades, to effectively detect system anomalies based on heterogeneous monitoring data via cross-modal attention.
- Extensive experiments on simulated data and datasets from Huawei Cloud demonstrate the effectiveness of Hades, as well as the contribution of each design to Hades.
- We collect an annotated dataset containing complex log semantics and metric patterns, which is released with our code for this study [18] of this work to facilitate other related practitioners and researchers.

II. Problem Statement

We first introduce essential terminologies. A *log message* is a line of the standard output of logging statements, composed of constant strings (written by the developers) and variable values (determined by the system) [19]. Parsing a log message is to remove all variables to obtain a *log event*, which describes system run-time events [20]. Log messages chronologically collected within a certain period constitute a *log sequence*. Figure 1 shows an example of logs and obtained log events. On the other hand, *metrics* are the numerical measurement of system performance that are sampled uniformly. Consecutive points within a certain period make up a *metric segment*. By collecting both the logs and metrics in a given period of length T, we obtain a *chuck* with time-aligned heterogeneous data. The value of T is determined according to real-world requirements. *Anomalies* are abnormal system behaviors, events, or observations that do not conform to the expected patterns in the run-time information [20]. These anomalies often indicate system issues and could evolve into errors or failures.

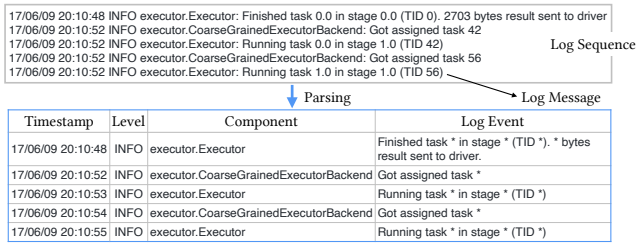

Fig. 1: Examples of logs and parsed logs.

We formalize the heterogeneous anomaly detection task and present this learning problem with inputs and outputs. Given a data chunk, we aim to determine the current system status as abnormal or normal, denoted as 1 and 0, respectively. Let $\langle \boldsymbol{X}_{1:N}, Y_{1:N} \rangle = \{(\boldsymbol{X}_1, y_1), (\boldsymbol{X}_2, y_2), \cdots, (\boldsymbol{X}_N, y_N)\}$ be the training data chunks with corresponding status labels, where $\boldsymbol{X}_i = (\boldsymbol{X}_i^{\boldsymbol{l}}, \boldsymbol{X}_i^{\boldsymbol{m}}), (i = 1, 2, ..., N)$ is the i-th data chunk, and $y_i \in \{0, 1\}$ denotes the status. In the i-th chunk, $\boldsymbol{X}_{i,1:L}^{\boldsymbol{l}} = [\boldsymbol{x}_{i,1}^{\boldsymbol{l}}, \boldsymbol{x}_{i,2}^{\boldsymbol{l}}, ..., \boldsymbol{x}_{i,L}^{\boldsymbol{l}}]$ denotes the log sequence, where L is the number of log events generated during the period of length T; the metric segment is denoted by $\boldsymbol{X}_{i,1:T}^{\boldsymbol{m}} = [\boldsymbol{x}_{i,1}^{\boldsymbol{m}}, \boldsymbol{x}_{i,2}^{\boldsymbol{m}}, ..., \boldsymbol{x}_{i,T}^{\boldsymbol{m}}] \in \mathbb{R}^{T \times M}$, where M and T are the number and length of monitoring metrics, respectively. The goal is to model the relations behind $\langle \boldsymbol{X}_{1:N}, Y_{1:N} \rangle$, and then for each incoming unseen instance X_{N+1}, we can predict the status y_{N+1}.

III. Data Collection

In practice, engineers usually analyze different sources of system run-time information for troubleshooting. However, manual inspection is tedious and fallible, especially when facing massive data. To explore the opportunity to automate this process, we systematically study how system anomalies affect the different types of monitoring data. Moreover, most industrial datasets are access-restricted, and the publicly accessible data are often too small or single-source due to security and privacy concerns. To alleviate this problem, we have released our collected dataset on [18]. It contains large-scale logs and metrics generated from a distributed computing system, which underpins our study and will facilitate the advancement and openness of the community.

Our data collection comprises four steps: 1) deploy the infrastructure, 2) conduct workloads to generate monitoring data, 3) inject typical faults to simulate industrial production anomalies, and 4) collect heterogeneous monitoring data simultaneously. We describe them in detail as follows.

A. Data Generation

Apache Spark is a widely-used framework for big data processing [21]. We deploy Spark 3.0.3 on a distributed systems cluster containing a master node and five worker nodes (virtual nodes supported by Docker [22]) in our laboratory environment. To conduct workloads, we employ a big data benchmark suite, HiBench [23]. Unlike existing collections (e.g., [13], [24], [25]) running only word counting, we perform 19 kinds of workloads covering diverse service application scenarios. The workloads fall into five categories and are diverse in terms of resource usage, including CPU, I/O, memory, and network. For example, the workload random forest performs complex computation, and the workload word counting requires transaction-intensive file inputting. Details about the settings of data collection are introduced in A.

We first repeat each workload without faults seven times, where the run-time data are named as *standard* data when no fault is injected during the entire workload. We mainly collect two types of run-time data, i.e., logs and metrics. Logs are aggregated by Spark automatically, and metrics are sampled (per second) and collected via an open-source monitoring tool [26]. In the following analysis, we focus on 11 monitoring metrics reflecting four critical aspects of the system status (i.e., CPU, I/O, memory, and network), such as CPU system usage, device read speed ("rkb/s"), memory usage, and network throughput rate.

B. Fault Injection

After gathering standard data, we inject 21 typical types of faults that are selected based on previous research [12], [24], [25] and our investigation of the typical service failures at Huawei Cloud. They fall into four categories:

- Process suspension: suspend a process for one minute, including the Application Worker, Master, Node Manager, Datanode, Resource Manager, and Namenode.
- Process killing: kill each process once, including the Application Worker, Master, Node Manager, and Datanode. Killing these processes does not terminate the running workload immediately.
- Resource stress: load the system by injecting CPU, memory, and other resource hogs, supported by [27].
- Network faults: inject Linux traffic faults such as losing packets, high latency, and flashing disconnections.

Each fault lasts for one minute, and the time from fault injection to fault clearance is called *fault duration*. The data collected for the rest of the time are regarded as *fault-free*.

C. Data Annotation

We invite two Ph.D. students experienced in software reliability as annotators. They are unaware of when the faults are injected or cleaned, so the labeling is completely based on the data without off-site information. The principle to label an anomaly is that the in-process data manifest discrepancies from the standard data, e.g., error logs and unusual metric jitters. Meanwhile, the abnormal manifestations should align with the expected impact of the injected fault, which is manually checked by annotators. We do not simply regard all data generated during the fault duration as abnormal because many faults can be immediately tolerated by the system's fault-tolerant mechanisms, incurring no anomalies to the system. In this case, we treat the corresponding data as normal. The label is accepted if the two students give the same label for one chunk independently. Otherwise, they will discuss with a post-doc until reaching a consensus. The inter-annotator agreement [28] (i.e., a measure of the reliability of an annotation process) achieves 0.876 before adjudication.

IV. Motivation

This section elaborates on our motivation by answering the following three questions:

- How do logs manifest system anomalies?
- How do metrics manifest system anomalies?
- How do monitoring data (including logs and metrics) reflect the system status?

A. How do logs manifest system anomalies?

Logs may not be susceptible enough to some system faults. Only 3.62% of positively labeled chunks are anomalous from the log's perspective. A typical example provides a closer look. If the network drops some packets, the service response becomes slow but may not hit the pre-defined timeout threshold, and thereby no anomalous log event will be reported. The main reason lies in the inherent deficiency of logs. Logging [20], [29]–[31] is a human activity heavily relying on developers' knowledge. While it is relatively easy to write logs describing severe failures, subtle performance issues, e.g., gray failures [32], are often hard to identify. Thus, logs capturing such anomalies could be missing.

Digging into logs, we observe that the lexical semantics are noteworthy. Specifically, the appearance of some log tokens indicates anomalies. These tokens only occur in abnormal

sequences, and their semantics describe unusual system events, e.g., "uncaught" and "exception" in the event "Uncaught exception in thread". This observation is in line with the programming habits of developers, as well as previous studies [7], [33], [34]. It also validates that log semantics can reflect the current system status to some extent.

Moreover, the contexts of logs are crucial for detecting anomalies since logs carry the information of program control flows [35]. For example, a normal sequence of log events is an event reporting the "final application" state followed by the log event "Shutdown hook called". When anomalies happen, the event "Shutdown hook called" may occur before reporting the "final application" state. In this case, the application state will be regarded as undefined or failed because the master has not received the message informing the application state. Hence, the contextual semantics of logs also contain information that indicates whether the system is healthy.

Finding 1: Logs sometimes cannot record fine-grained information and therefore, cannot manifest all system anomalies. Moreover, both lexical and contextual semantics are important with respect to reflecting the system status.

B. How do metrics manifest system anomalies?

Metrics are responsive to anomalies by continuously recording system status. However, there are still some anomalous periods ignored by many existing metric-based detectors.

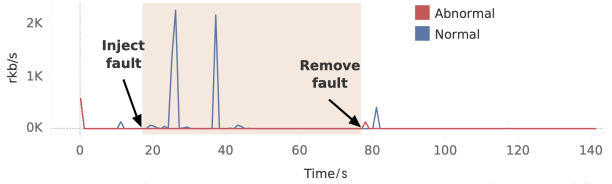

Fig. 2: Suspending the Datanode incurs anomalies manifested in the metric "rkb/s", but no novel pattern exists.

Existing anomaly detectors usually try to identify novel metric patterns through a comparison with normal system behaviors, i.e., novelty detection [36]. They mainly focus on local patterns (e.g., spikes, level shifts) rather than global patterns spanning the entire workload. However, system faults are not necessarily manifested by local novel patterns. Figure 2 displays an example that the metric "rkb/s" during the fault "Datanode suspension", as shown by the red line. The metric "rkb/s" is abnormal as a whole, but its local patterns are hard to identify as anomalies. Specifically, the red line remains zero after fault injection, which is unexpected as there should have been I/O activities going on. The blue line depicts the normal (standard) status as a comparison. However, the blue line also remains zero after the 90-*th* second because the system does not exchange data near the end of the workload. So "rkb/s" in either the normal or abnormal status can stay at zero for a while. This indicates that a metric can behave very similarly when the system is in the opposite status. Such patterns (which may reflect the opposite system statuses) will confuse most existing methods relying on novel pattern mining, leading to performance degradation. The inherent reason is that system metrics cannot completely reflect the software's inner execution logic. Fortunately, such anomalies can be detected by referring to the logs, where suspicious events like *Exception in createBlockOutputStream* are reported.

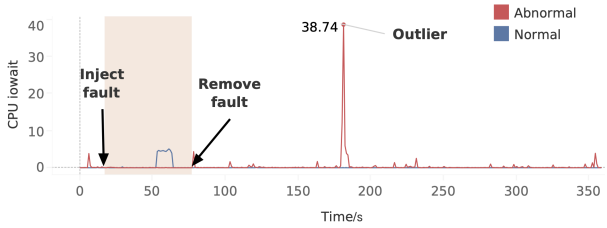

Fig. 3: An acceptable outlier of "CPU iowait" in a fault-free period may trigger a false alarm by most automated tools.

On the flip side, unusual metric fluctuations may trigger alarms even when no anomaly exists currently. Among all metric segments manually labeled as positive, 8.87% of them are collected in fault-free periods, indicating that relying solely on metrics may cause false alarms due to the overreactions of metrics. Figure 3 displays an example that the metric "CPU iowait" generates a rare heartbeat spike even in the fault-free period. Such sporadic and transient fluctuation is acceptable without affecting the service, thereby no alarm should be triggered. This case suggests that other information should be involved to mitigate the issues caused by the over-sensitivity of metrics to avoid unnecessary engineering resource waste.

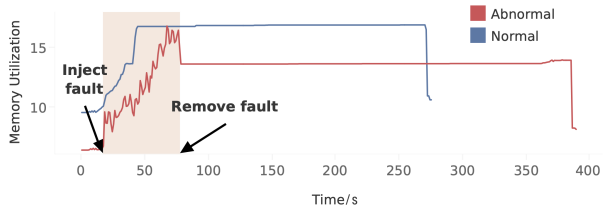

Fig. 4: The irregular metric jitters cannot be detected by single point-based detectors.

In addition, we find that metric patterns presented at the segment level (i.e., a series of continuous metric points) sketch issues much better than single-point outliers. For example, the metric "memory usage" performs noticeably abnormal jitters when injecting a "virtual memory hog" (Figure 4). However, these jittering points are not outliers because their values are not extraordinarily high or low. Clearly, the outlier perspective ignores high-order data variations, such as the scope and denseness, resulting in missing alarm issues.

Finding 2: Metrics are responsive to system status changes but still insufficient in many cases. Their over-sensitivity may cause false alarms on uncommon yet acceptable fluctuations. Besides, segment-level metric patterns can be more useful in anomaly detection than single-point outliers.

C. How do monitoring data reflect the system status?

We summarize representative faults and their effects on logs and metrics in Table I, showing that faults can affect logs

TABLE I: Typical faults and the corresponding anomalous manifestations of logs and metrics

Faults	Anomalies in logs	Anomalies in metrics
Memory hog	Warnings (reaches the memory limit)	Memory-related metrics rise steeply
Virtual memory hog	Errors (reporter thread fails)	CPU and memory-related metrics jitter
I/O hog	Warnings (slow ReadProcessor)	I/O-related metrics rise steeply
Network delay	Warnings (executor heartbeat timeout)	Network-related metrics suddenly drop
Connection flash	Nothing **(silent)**	Network-related metrics suddenly drop and quickly restore
Datanode killed	Errors (excluding datanode)	Related metrics plummet to zero **(silent)**
Secondary namenode killed	Errors (failed to connect to <IP>)	Related metrics plummet to zero **(silent)**

and metrics simultaneously. For example, one resource hog can incur both sudden spikes in metrics and warning logs for limited resources. Moreover, some cases see an evident complementary relationship between logs and metrics. When the Datanode or Secondary Namenode is killed, related metrics plummet to zero but cannot be detected (i.e., silent) since these metrics also plummet to zero if the application ends normally. Metric-based anomaly detectors cannot distinguish such abnormal drops from normal ones. In this case, logs play the role of additional information to help determine the system status. On the contrary, when the connection between nodes flashes, no warnings or errors are generated in logs since the network disconnection time is too short to affect program operation. Nevertheless, network-related metrics faithfully reflect such transient anomalies, e.g., the network throughout that drops rapidly during flashes. Such observations align with intuition. Basically, logs record the software's internal execution logic, while metrics provide an external view by measuring software services' performances, resource usage, etc. Thus, combining the two can better portray the system status due to their collaborative and complementary relationships.

Furthermore, a fault can affect logs and metrics to varying degrees. For example, killing the Datanode during the Latent Dirichlet Allocation (LDA) application causes 29 abnormal metric segments while only one log sequence reports anomalies. Another example is that when suspending the Namenode in word counting, the related metrics experience a sharp drop and remain unchanged since most of the computation has been done. Yet tens of logs reporting a failed state are generated because the worker nodes keep sending warnings while the master node cannot receive messages. Hence, simply regarding all types of data equally important is unreasonable since the more severely affected part may deserve more attention. In brief, we should combine and assign appropriate weights to metrics and logs to promote effective anomaly detection.

Finding 3: Metrics and logs can both respond to anomalies, but neither is sufficiently informative. They have collaborative and complementary relationships in providing clues for the system's health. Also, the degree to which they are affected by the same anomaly can vary greatly.

These findings support our motivation to develop an automated anomaly detector based on heterogeneous monitoring data, i.e., logs and metrics. This results in HADES, our solution to attack the previously mentioned challenges.

V. Methodology

Figure 5 presents the overview of Hades, a heterogeneous anomaly detector for software systems via cross-modal attentive learning. It consists of four components: Log Modeling, Metric Modeling, Heterogeneous Representation Fusion, and Detection. It is trained by both labeled and pseudo-labeled data in a Semi-supervised Training manner. Hades aims to infer the system status from current heterogeneous monitoring data based on historically extracted patterns. We incorporate domain knowledge and the insights obtained from our previous study to design a practical model architecture. Specifically, for logs, Hades captures lexical and contextual log semantics and maps each raw log sequence into a low-dimensional representation. For metrics, Hades preserves aspect-aware information at the segment level along the timeline and learns cross-aspect correlations. Our framework also employs an attention-based fusion module with cross-modal learning to acquire a global representation, which is fed into a successive detection component. Consequently, it will trigger an alarm to operations engineers when a noteworthy anomaly occurs.

A. Log Modeling

Log modeling contains three steps: log parsing, log vectorization, and log representation learning. It extracts meaningful log features including lexical and contextual semantics.

1) Log Parsing: In this step, we transform unstructured log messages into structured log events. As aforementioned, raw log messages are unstructured and contain variables that can hinder log analysis [19]. Therefore, we first employ a widely-used parser Drain [37] to extract log events since it has shown effectiveness and efficiency in the previous evaluation study [38]. Next, we conduct a stable sorting based on the log timestamps. As a result, all valid log messages are transferred into chronologically arranged log events.

2) Log Vectorization: This phase turns textual log events into semantics-aware numerical vectors. We utilize FastText [15] to capture the intrinsic relationships of log vocabulary and preserve the important log semantics. FastText is a popular, lightweight, and efficient technique for producing word embeddings that can represent semantic similarities between words. After training, FastText maps every token into a E-dimension vector, so a log event $\boldsymbol{x^l}$ is transformed into a token embedding list $\boldsymbol{V} = \{v_i\}_{i=1}^{\omega} \in \mathbb{R}^{\omega \times E}$, where ω is the token number of an event. Subsequently, we average all elements inside $\boldsymbol{V}$ to acquire a sentence embedding

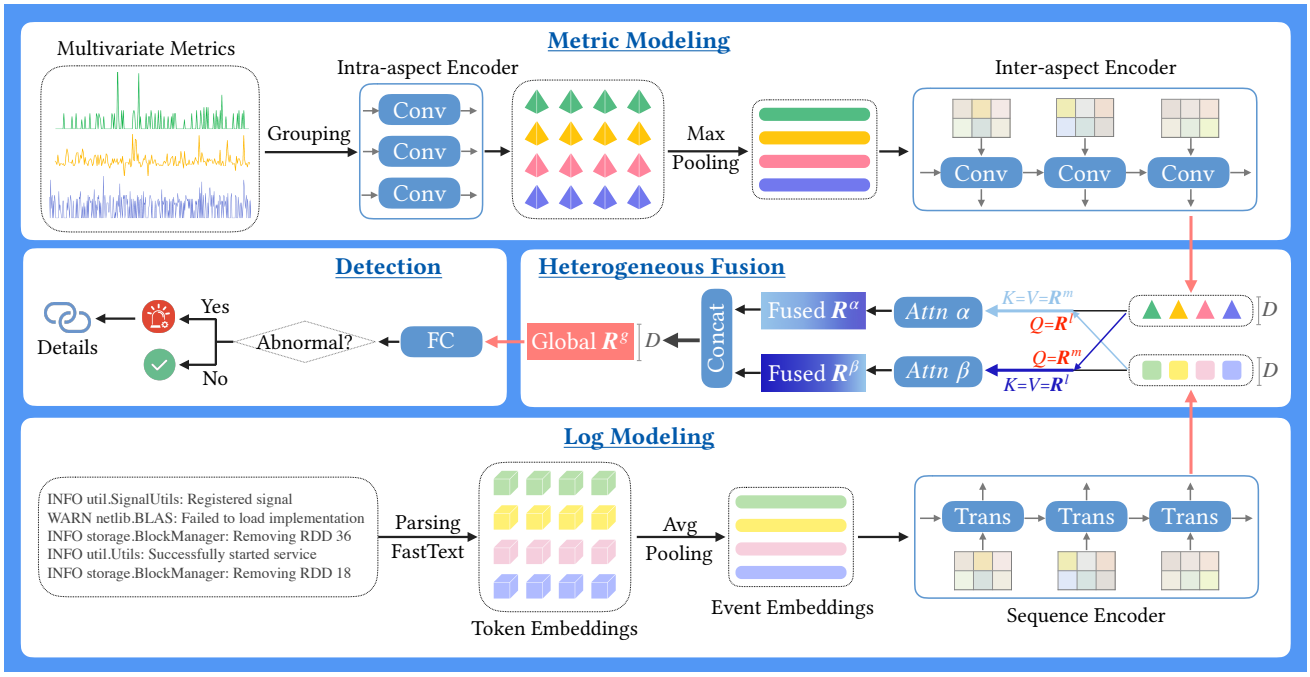

Fig. 5: Overview of Hades.

$\bar{\boldsymbol{V}} = \frac{1}{\omega} \times \sum_{i=1}^{\omega} \boldsymbol{v}_i$. Consequently, a log sequence $\boldsymbol{X}^{\boldsymbol{l}}_{1:L}$ can be denoted by a sentence embedding list $\bar{\boldsymbol{V}} = \{\bar{\boldsymbol{V}}_i\}_{i=1}^{L} \in \mathbb{R}^{L \times E}$.

3) Log Representation Learning: This step models the log contextual semantics and generates log representations with learned information. In particular, the sentence embeddings of a sequence obtained from the previous phase are fed into a sequence encoder, which is composed of two Transformer encoder layers [16]. This encoder captures contextual dependencies across the events. Afterward, a fully-connected (FC) layer maps the output into a D-dimensional feature space. Hence, we obtain the log representation of a chunk, denoted by $\boldsymbol{R}^{\boldsymbol{l}} \in \mathbb{R}^{L \times D}$. If the log sequence is too long, we partition it into non-overlapping fixed-size sub-sequences and conduct the above steps. For a too-short sequence, we pad it with zeros.

B. Metric Modeling

Metrics are modeled hierarchically in this section with respect to segment-level patterns motivated by the second finding in § IV-B. The module comprises an intra-aspect encoder and an inter-aspect encoder (also shown in Figure 5).

The rationale behind the design is three-fold: (1) Metrics sketching the same aspect of the system should be modeled together. Monitoring metrics can reflect various aspects of system performance, e.g., CPU utilization, memory utilization, etc. Generally, metric patterns of the same aspect share certain similarities (e.g., CPU user usage and CPU system usage both characterize CPU usage). Such metrics should be grouped and regarded as multi-variate time series (MTS) to be analyzed together. (2) Metrics depicting different aspects should be modeled separately. If two metrics belong to different aspects, their patterns can be very different. For example, the disk usage tends to be stable while the I/O throughput may fluctuate violently even under the normal status. So metrics of different aspects should be fed into separate models to capture fine-grained information. (3) While metrics of different aspects tend to develop distinct patterns, they still exhibit some inter-aspect correlations when anomalies occur. For instance, if a worker node loses connection with the master node, many metrics such as the CPU utilization and route cost will drop precipitously and stay at zero since data exchange or computation is interrupted.

Based on (1) and (2), we propose an intra-aspect encoder to capture the aspect-oriented temporal dependencies and cross-metric relationships. (3) inspire us to design an inter-aspect encoder to learn correlations across aspects integrally. The two encoders learn the aspect-aware representations hierarchically.

1) Intra-Aspect Encoder: As metrics should be modeled in an aspect-aware manner, that is, modeling metrics of the same aspect together while modeling metrics of different aspects separately, the internal model must be computationally efficient. Thus, we adopt 1D causal convolution [17]. Conventional convolution networks face the problem of information leakage (i.e., the output depends on future inputs) and the inability of sequential dependency modeling. Causal convolution is designed to mitigate these limitations, which meets our needs by being parallelizable, lightweight, and accurate [39]. This phase decomposes the metrics into γ groups according to their corresponding aspects based on our domain knowledge. After that, metrics of the same aspect are taken as an MTS and fed into a separate intra-aspect encoder composed of a multi-layer causal convolution network. After appropriate padding and chomping, the γ intra-aspect encoders output γ feature vectors $\boldsymbol{h}^{\boldsymbol{m}}$. Finally, we conduct a max-pooling operation on the feature dimension and stack the outputs to form a latent feature vector $\boldsymbol{H}^{\boldsymbol{m}} \in \mathbb{R}^{T \times \gamma}$.

2) Inter-Aspect Encoder: This module also leverages causal convolution to learn the inter-aspect features. Such structure helps model complex patterns by capturing multi-level information. We take $\boldsymbol{H^m}$ outputted by the intra-aspect encoder as an MTS with $T \times \gamma$ series and feed it into the inter-aspect encoder to model the correlations between metric aspects. In this way, the metrics $\boldsymbol{X^m_{1:T}}$ inside a chunk are embedded into a D-dimensional representation, denoted by $\boldsymbol{R^m} \in \mathbb{R}^{T \times D}$. Note that data of all modalities should be embedded into the same D-dimensional feature space for alignment.

C. Heterogeneous Representation Fusion

We design a novel cross-modal attention mechanism to fuse heterogeneous representations and bridge the gap between logs and metrics. Previous phases embed logs and metrics into a D-dimensional feature space. These representations are fed together into this fusion module, defined by two attention layers [16]. The first one (Attn-α) takes the log representation $\boldsymbol{R^l}$ as the Query while the metric representation $\boldsymbol{R^m}$ as the Key and Value. It matches the log events explaining the metric changes. Symmetrically, in the second attention layer (Attn-β), $\boldsymbol{R^m}$ plays the role of the Query, and $\boldsymbol{R^l}$ serves as the Key and Value. It helps to find the performance variations aligned with log contents to enhance log expressiveness. Mathematically, given the Query, Key, and Value, we calculate:

$$\mathsf{Fuse}(Q, K, V) = \tanh\left(\left[\mathsf{softmax}(QW_s K^{\mathrm{T}})V; Q\right] W_a\right) \quad (1)$$

where W_a and W_s are learnable parameters; $[\cdot;\cdot]$ denotes concatenation. Afterward, outputs from Attn-α and Attn-β are concatenated inside the D-dimensional space to constitute a global representation $\boldsymbol{R^g} \in \mathbb{R}^{(T+L)\times D}$, defined as:

$$\boldsymbol{R^g} = [\mathsf{Fuse}(\boldsymbol{R^l}, \boldsymbol{R^m}, \boldsymbol{R^m}); \mathsf{Fuse}(\boldsymbol{R^m}, \boldsymbol{R^l}, \boldsymbol{R^l})] \quad (2)$$

Above all, switching the roles of the two modalities allows devoting more attention to the features of different modalities that convey similar information, which is more likely to be responsive to changes in the system status. Also, it can retain meaningful intra-modal patterns explicitly by directly concatenating the Query with the attended Value. In this way, the global representation reserves not only the shared information and cross-modal interactions but also the salient intra-modal dependencies and the inferred features due to the complementary relationship between logs and metrics.

D. Detection

Finally, we feed the representation $\boldsymbol{R^g}$ into stacked FC layers followed by a softmax layer. The output $\hat{y} \in \{0, 1\}$ represents the status being normal or abnormal, computed by:

$$\hat{y} = \mathsf{argmax}[\mathsf{softmax}(U\sigma(V \cdot \boldsymbol{R^g} + b) + c)] \quad (3)$$

where U and V are learnable weight matrices; b and c are bias terms; $\sigma(\cdot)$ is the ReLU activation function [40]. This module generates an alarm upon detecting an anomaly. To facilitate further analysis by engineers, we also provide a visual interface that enables a convenient review of logs and metrics of the suspicious chunk, as shown in Figure 6.

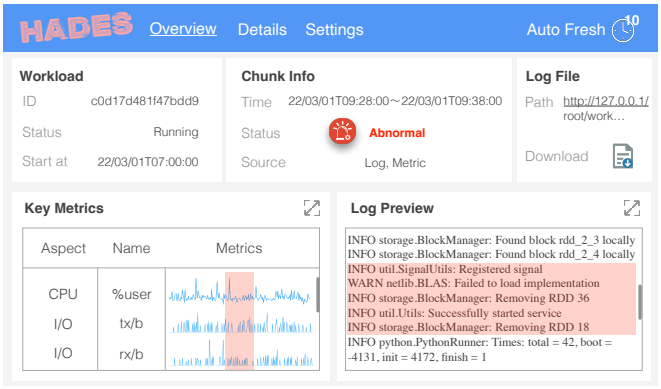

Fig. 6: A demo for reviewing the detected chunk, where the light red rectangle represents the currently focused window.

E. Semi-supervised Training

To reduce the cost of manual labeling and leverage human expertise simultaneously, we apply semi-supervised learning to train our model. Semi-supervised learning leverages a small amount of labeled data and unlabeled data for training, based on the smoothness assumption: a normal sample should be closer to another normal sample rather than an abnormal sample in the feature space, and vice versa. Previous studies adopt the assumptions that logs with similar semantics should share the same detection indicator [8], and metrics with similar patterns lead to the same existence indicator of performance issues [4]. Intuitively, heterogeneous data with similar log semantics, metric patterns, and cross-modal interactions are likely to share the same label. With this intuition, we devise a two-fold semi-supervised training strategy, shown in Figure 7.

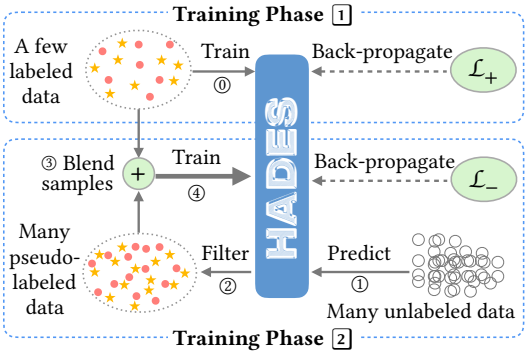

Fig. 7: The semi-supervised training of Hades.

We first label data generated during frequent faults by representative workloads (accounting for about 10% of the whole data). The labeled data are used to train Hades in a supervised manner in the first phase by minimizing the binary cross-entropy loss (denoted as $\mathcal{L}_+$) for several epochs. Next, we generate predictions of the remaining unlabeled data and filter in predictions with high confidence as pseudo labels. As the model is a probabilistic classifier in nature and thereby provides confidence scores directly, we take the maximum of the outputted classification probabilities as the corresponding prediction's confidence. The second phase blends pseudo-labeled and labeled data to train the framework, attempting to

minimize the following loss function:$\mathcal{L} = (1-\lambda)\cdot\mathcal{L}_+ + \lambda\mathcal{L}_-$, where $\mathcal{L}_-$ is the binary cross-entropy loss of the pseudo-labeled data, and λ is a hyper-parameter ranging $[0, 1]$.

VI. Evaluation

We evaluate Hades by answering the following research questions (RQs):

- **RQ1**: How effective is Hades in anomaly detection?
- **RQ2**: What is the contribution of each design of Hades?
- **RQ3**: How sensitive is Hades to the length of a chunk?

A. Experiment Setup

1) Datasets: Besides the in-lab Dataset $\mathcal{A}$ (§ III), we also evaluate Hades on other two datasets ($\mathcal{B}$ and $\mathcal{C}$) containing heterogeneous monitoring data from two different industrial cloud services of Huawei Cloud. Dataset $\mathcal{B}$ and $\mathcal{C}$ contain eight types of faults respectively: CPU stress, memory stress, high disk I/O latency, disk partition full, network flashing, long network latency, high packet loss rate, and zombie process. Metrics are sampled once per minute. Drain [37] extracts 72 and 104 log events from Dataset $\mathcal{B}$ and $\mathcal{C}$, respectively.

Chunks are obtained via a timestamp-based sliding partitioning strategy [33]. For all three datasets, we split the first 70% generated data as the training set and the last 30% as the test set, meaning that data in different sets are produced in different periods (also different workloads for Dataset $\mathcal{A}$). Inside the training set, we choose five representative workloads out of 19 in $\mathcal{A}$ (choose one in each category according to [23]) and seven faults out of 21 for labeled training data, that is, only annotations of the chosen workloads during the standard status or the selected faults' duration are adapted for training. The rest of the training set is taken as unlabeled training data. For Dataset $\mathcal{B}$ and $\mathcal{C}$, we invite three experienced reliability engineers to annotate the test set, and training data with two selected faults (memory stress and disk partition full), following a similar procedure described in § III-C. Few disagreements occurred as anomalies are well-defined by the engineers who are familiar with their system.

In this way, chunks belonging to the same faulty or non-faulty period will exist either in the training/test set so as to avoid possible data leakage caused by random split. Besides, the test set contains unseen anomalies incurred by non-selected faults, so we can evaluate the model's ability to infer new abnormal patterns, considering knowing all the anomaly patterns in advance is usually impossible. Table II shows the statistics of our datasets.

TABLE II: Dataset statistics

Dataset	Log Messages	Metric Length	Manually Labeling
$\mathcal{A}$	1,435,139	64,422	10.87%
$\mathcal{B}$	76,724	7,473	13.70%
$\mathcal{C}$	1,148,563	15,945	11.24%

2) Baselines: We compare Hades with ten baselines in different training manners, including unsupervised, semi-supervised, and supervised ones. In particular, despite being specially designed for bad software identification, SCWarn [41] (unsupervised) is the only multi-source approach for binary classification in the software engineering literature, as far as we know. Six state-of-the-art single-source baselines are adopted: Deeplog [9] (unsupervised), PLELog [8] (semi-supervised), and LogRobust [7] (supervised) are log-based. Omninomaly [6] (unsupervised), Adsketch [4] (semi-supervised), SRCNN (unsupervised) and its supervised variant SRCNN-s [5] are metric-based. We also adopt supervised log-based SVM (SVM-$\mathcal{L}$) and metric-based SVM (SVM-$\mathcal{M}$) [42] as the representation of traditional techniques.

3) Evaluation Measurements: As we tackle anomaly detection in a binary classification manner, we adopt the widely-used measurements to gauge models' performances: *Rec*= $\frac{TP}{TP+FN}$, *Pre*= $\frac{TP}{TP+FP}$, *F1*= $\frac{2\cdot TP}{2\cdot TP+FN+FP}$, where *TP* is the number of successfully detected abnormal chunks; *FP* is the number of normal chunks incorrectly triggering alarms; *FN* is the number of undetected abnormal chunks.

4) Implementations: Our code for implementing Hades is publicly available at [18]. The log encoder adopts a hidden size of 1024 for four layers. We use Gensim [43] to train 32-dimensional word embeddings. The intra-aspect metric encoder comprises two layers, and the inter-aspect encoder has three layers with a hidden size of 256. The decoder consists of four layers with a hidden size of 512. We use the Adam optimizer [44] with an initial learning rate of 0.001. The batch size is 128, and the epoch for each training phase is 50.

As for baselines, we adopt the public implementations of [4]–[6], [8], [41], and an open-source toolkit [33] to implement [7], [9], whose original paper did not provide code. We determine the hyper-parameter combination achieving the highest test *F1* for each baseline. All experiments are conducted on a single NVIDIA GeForce GTX 1080 GPU.

B. RQ1: Overall Performance of Hades

Table III presents the overall performance comparison. Each result of Hades is averaged over three independent runs with three random seeds and the values of evaluation measurements vary in very small decimal cases. Hades outperforms all baselines by a significant margin, achieving the best result on every evaluation measurement. Specifically, its *F1* scores are 0.864, 0.975, and 0.960, 9.12%$\sim$174.41% higher than competitors on average. The high *Rec* and *Pre* scores of Hades illustrate that there are very few missed anomalies or false alarms. Thus, we can argue that Hades is considerably effective to detect system anomalies, redounding to economizing engineering resources.

Compared with SCWarn, the success of Hades boils down to three aspects: (1) Hades adopts semi-supervised learning to balance annotation cost and human oversight. SCWarn is unsupervised and detects anomalies based on the next timestamp prediction, yet accurately predicting is really difficult in a large-scale system with complex behavior patterns. Sometimes software behaviors are inherently unpredictable [45]. (2) Hades extracts multi-level log semantics and represents log events via succinct and low-dimensional embeddings. In comparison, SCWarn transforms logs into event occurrence sequences, generating an over-large sparse feature matrix for

TABLE III: Overall Performance Comparison.

Models	Dataset $\mathcal{A}$			Dataset $\mathcal{B}$			Dataset $\mathcal{C}$		
	F1	*Rec*	*Pre*	*F1*	*Rec*	*Pre*	*F1*	*Rec*	*Pre*
SCWarn	0.321	0.389	0.273	0.497	0.643	0.405	0.491	0.585	0.423
SVM-$\mathcal{L}$	0.289	0.707	0.181	0.541	0.756	0.421	0.481	0.742	0.356
DeepLog	0.259	0.386	0.194	0.386	0.526	0.305	0.375	0.524	0.292
PLELog	0.314	0.602	0.213	0.463	0.618	0.371	0.434	0.597	0.341
LogRobust	0.404	0.684	0.287	0.524	0.718	0.413	0.495	0.698	0.383
SVM-$\mathcal{M}$	0.536	0.833	0.395	0.608	0.839	0.477	0.556	0.801	0.426
OmniAnomaly	0.681	0.788	0.601	0.827	0.863	0.794	0.812	0.896	0.743
Adsketch	0.404	0.476	0.351	0.543	0.644	0.470	0.538	0.649	0.459
SRCNN	0.342	0.614	0.237	0.467	0.701	0.350	0.472	0.586	0.394
SRCNN-s	0.784	0.826	0.745	0.898	0.938	0.861	0.883	0.926	0.844
Hades	**0.864**	**0.870**	**0.859**	**0.975**	**0.984**	**0.966**	**0.960**	**0.969**	**0.951**

log events and discarding log semantics. The sparse matrix poses barriers to extracting meaningful features, especially when hundreds of events exist. (3) Hades devises an attentive fusion to capture significant cross-modal interactions and bridge temporal and textual representations. SCWarn simply concatenates the representations and ignores the vast gap between metrics and logs, i.e., the information form and the input size discrepancies.

The superiority of Hades over single-modal baselines stems from the effective use of logs and metrics. Compared with the best single-modal model (SRCNN-s, a supervised approach), Hades increases *F1*, *Rec* and *Pre* by 9.12%, 4.91%, 13.22% on average, respectively. Such improvement is exciting and reasonable. Our previous study reveals that metrics and logs can both reflect anomalies, and neither of them is sufficient (§ IV-C). However, these baselines only review one data source and omit important information hidden in the other source, so they suffer performance degradation, especially for those without knowledge of historical anomalies.

In addition, training an epoch takes Hades two minutes on average while baselines take 0.37∼9.01 minutes per epoch. The prediction delay is negligible as the prediction of all mentioned approaches is less than 0.1s for a chunk. The efficiency of Hades with nearly real-time detection and acceptable offline training time engages its industrial prospect.

In brief, Hades effectively detects system anomalies on all datasets. It significantly improves the effectiveness compared to all baselines concerning every evaluation measurement.

C. RQ2: Individual Contribution of Modules

1) Derived Models: We conduct an extensive ablation study on Hades. Particularly, we derive seven models based on the original Hades to investigate the contribution of the introduction of heterogeneous information, representation fusion, cross-modal attention, and intra-modal feature extraction.

- Heterogeneous Information: Hades *w/o*$\mathcal{M}$ removes metrics, containing a log encoder (duplicated from § V-A) and a decoder. Similarly, Hades *w/o*$\mathcal{L}$ removes logs, containing a metric modeling module (duplicated from § V-B) and a decoder. That is, representations of logs and metrics are directly fed into the respective decoder separately.
- Representation fusion: Hades *w/o*$\mathcal{F}$ performs a Boolean `OR` operation on the results from Hades *w/o*$\mathcal{M}$ and Hades *w/o*$\mathcal{L}$. It is built on the motivation that it is natural to process each type of data individually and then aggregate results instead of fusing representations as Hades does.
- Cross-modal attention: Hades *w/o*$\mathcal{A}$ removes the cross-modal attention and simply concatenates the representations of metrics and logs as the global representation. Hades *w/o*$\mathcal{C}$ operates conventional self-attention on the two representations separately and then concatenates them, rather than using cross-modal attention. Other modules except for the fusion module (§ V-C) are the same as Hades.
- Intra-modal feature extraction: Hades *w/o*$\mathcal{H}$ is designed for validating the contribution of the hierarchical aspect-aware metric encoder (§ V-B), which models metrics in an aspect-agnostic manner based on causal convolutions by encoding all metrics simultaneously. Hades *w/o*$\mathcal{S}$ aims to present the usefulness of log lexical semantics by replacing the word embedding with one-hot encoding.
- Semi-supervised training: Hades-Anno are trained by annotated data rather than leveraging the semi-supervised training approach, which is used to evaluate the effectiveness gap between our proposed semi-supervised training and supervised training.

2) Experimental Results: Table IV shows the experimental results, underpinning four key conclusions: (1) Introducing heterogeneous information contributes incredibly to enhancing anomaly detection in view of two observations. Hades outperforms Hades *w/o*$\mathcal{M}$ and Hades *w/o*$\mathcal{L}$ considerably, especially in *Pre*, indicating that Hades can reduce a large number of false alarms, thereby increasing *F1* by 136.80% and 17.06% on average, respectively. Also, all heterogeneous data-based derived models outperform homogeneous variants (Hades *w/o*$\mathcal{M}$ and Hades *w/o*$\mathcal{L}$), further confirming that heterogeneous data goes far toward fully characterizing the system health. (2) Two shreds of evidence highlight the benefits of representation fusion: 1) Hades performs better than Hades *w/o*$\mathcal{F}$ (6.63% higher in *F1* on average); 2) the variants using representation fusion (Hades *w/o*$\mathcal{A}$ and Hades *w/o*$\mathcal{C}$) outperform Hades *w/o*$\mathcal{F}$. It is not surprising since Hades *w/o*$\mathcal{F}$ cannot fully mine cross-modal interactions. For example, the over-sensitivity of

TABLE IV: Experimental Results of the Ablation Study.

Models	Dataset $\mathcal{A}$			Dataset $\mathcal{B}$			Dataset $\mathcal{C}$		
	F1	*Rec*	*Pre*	*F1*	*Rec*	*Pre*	*F1*	*Rec*	*Pre*
Hades *w/o*$\mathcal{M}$	0.296	0.737	0.185	0.468	0.660	0.363	0.418	0.597	0.318
Hades *w/o*$\mathcal{L}$	0.719	0.718	0.720	0.840	0.886	0.799	0.832	0.888	0.782
Hades *w/o*$\mathcal{F}$	0.817	0.761	0.881	0.910	0.931	0.890	0.898	0.886	0.911
Hades *w/o*$\mathcal{A}$	0.829	0.831	0.828	0.943	0.966	0.921	0.928	0.931	0.926
Hades *w/o*$\mathcal{C}$	0.841	0.829	0.853	0.953	0.947	0.959	0.938	0.934	0.943
Hades *w/o*$\mathcal{H}$	0.852	**0.882**	0.824	0.967	0.980	0.955	0.952	**0.970**	0.935
Hades *w/o*$\mathcal{S}$	0.830	0.814	0.847	0.938	0.916	0.962	0.927	0.909	0.945
Hades-Anno	**0.866**	0.878	0.855	**0.979**	0.972	**0.986**	**0.961**	0.953	**0.970**
Hades	0.864	0.870	**0.859**	0.975	**0.984**	0.966	0.960	0.969	0.951

metrics (stated in § IV-B) may cause false alarms, and Hades *w/o*$\mathcal{F}$ fails to overcome such inaccuracy, while Hades alleviates this issue by utilizing logs as supplementary information to make more reasonable inferences. (3) Our cross-modal attention shows extraordinary value as Hades achieves better results than Hades *w/o*$\mathcal{A}$ and Hades *w/o*$\mathcal{C}$ because Hades can filter more informative features and exploit higher-order cross-modal interactions, thereby probing a stronger ability to characterize the system status. (4) The devised encoders make contributions via fuller- and finer-grained feature extraction, supported by the two observations: Hades *w/o*$\mathcal{H}$ generates more false alarms with mixing patterns of diverse aspects, resulting in overall worse performance (lower *F1*); Hades *w/o*$\mathcal{S}$ is relatively ineffective as it ignores crucial lexical semantics of logs. (5) The applied semi-supervised training is comparatively effective as supervised training while requiring only 10% labels. Hades is only 0.10%∼0.41% lower in *F1* than its supervised version. The success can be attributed to two reasons. First, only a small part of labeled data can cover various patterns, encouraging accurate inferences of Hades on similar unlabeled data samples. Second, we adopt a two-phase training strategy so the second semi-supervised training phase only considers pseudo labels with high confidence, thereby avoiding error accumulation during the two phases.

The results not only conform to the findings of our previous study in § IV, but also reveal the significance of heterogeneous data and the competence of our designs for intra- and inter-modal representation learning.

D. RQ3: Sensitivity to Chunk Length

The pre-determined chunk length T (e.g., the length of a chunk) may affect the framework's performance by affecting the dataset distribution. We herein evaluate the sensitivity of Hades to this hyper-parameter. We change the value of T while keeping all other hyper-parameters unchanged and conduct experiments as in RQ4. In detail, for Dataset $\mathcal{A}$, the default value of T is 10 (sec), ranging from 5 to 20; for Dataset $\mathcal{B}$ and $\mathcal{C}$, T is 5 (min) by default and ranges from 3 to 10, as the sampling frequencies of different datasets are different. Figure 8 presents the experimental results. Overall, Hades is fairly stable under different settings of T, further confirming its robustness. This makes Hades easy to deploy and launch in practice. We observe that more missing anomalies are rendered when the value of T deviates from the default configuration. This may be because the default granularity of chunks felicitously fits the anomalous patterns. Note that a larger T cannot guarantee an easier catch of anomalies. The influence of chunk length depends on multiple factors including model characteristics and system behaviors. In practice, we do not have to find the ideal T. It can be selected according to the validation result or the engineering expertise and requirements.

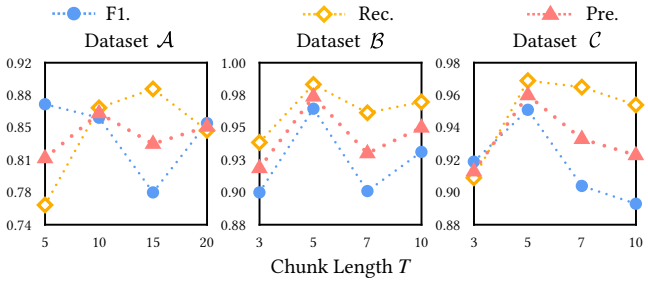

Fig. 8: The sensitivity to chunk length

VII. Discussion

A. Lessons Learned

We argue to be beware of the semantic gap between natural language and logs. Recent studies [7], [46], [47] have gradually adopted models pre-trained in natural language corpus to encode log texts. However, the semantics of natural language and logs are not exactly identical. For example, the word "successfully" usually expresses positive emotions in natural languages, yet it indicates an anomaly in the event "*Successfully connected to*<*IP*>:<*NUM*> for *BP*-*-<*>", which occurs when the Application Worker tries to reconnect with Datanode after losing the connection, and the re-connection succeeds. However, such a re-connection is not expected as connection loss should not happen. To mitigate this problem, Hades uses self-trained word embeddings and integrates multi-level semantics including word-level semantics, event-level semantics, and cross-event sequential dependencies.

B. Threats to Validity

A potential *internal threat* lies in the acquisition of log event embeddings. We use the average token embeddings as an event embedding, ignoring the sequential information inside a single event, as it is too time-consuming to extract the sequential dependencies event by event. Nevertheless, based on our study

and engineering experience, it is almost impossible for two log events to have identical tokens but in different orders, let alone to represent opposite system statuses. Thus, omitting the intra-event lexical order will not cause apparent adverse effects.

The *external threat* mainly comes from our datasets. Though Hades is evaluated on three datasets, it is yet unknown whether the effectiveness of Hades can be generalized across other datasets. To mitigate this threat, we use different datasets with representative workloads and typical faults to evaluate Hades, and the experimental results also demonstrate that Hades can work well even on unseen anomalies. We will also evaluate our approach based on more datasets in the future. In addition, the annotation process may introduce noise. Labeling principles vary depending on the system/person/purpose, sometimes causing inconsistency, especially when it comes to non-extreme anomalies or rare patterns, such as the slightly steep ups and downs in metrics, metric variations between normal fluctuations and abnormal jitters, log statements that rarely occur, etc. To alleviate this concern, we invite expert annotators that are familiar with the monitoring data. Semi-supervised training is also noise-resistant by only adopting a part of the labels. In practice, a company will share the definition of anomalies and develop standard labeling rules internally. It will also invite multiple professional practitioners to perform labeling to get labeling results with good consistency.

C. Limitations

We identify two limitations. First, Hades is complex and suitable for large-scale systems, such as cloud systems containing plenty of components rather than small systems with simple data patterns. Smaller systems may require only naive methods to obtain sufficiently accurate results, so applying Hades is not cost-effective. To improve its efficiency in small systems, we can distill the parameters of Hades to form a lightweight version. Second, Hades requires simultaneously collected logs and metrics. However, we find that in some large companies, metrics and logs are collected separately by different departments, and the sampling/logging frequency of different data sources varies dramatically. Sometimes a certain data source is even absent for a while. Thus, we add an extra mode to allow alternate use of homogeneous or heterogeneous anomaly detection. With either logs or metrics, practitioners can still take advantage of the superior feature extraction capability of Hades in the absence of a particular data source, thus extending the applicability of our approach.

VIII. Related Work

Many efforts have been devoted to automated anomaly detection for large-scale system reliability insurance based on logs and metrics [46]–[53]. Many advanced log-based anomaly detectors adopt deep learning to model log sequences. For example, [9] and [8] used recurrent neural networks to model normal logs and regard logs deviating from the model as abnormal. [7] tackled log instability by introducing the attention mechanism, as well as employing word embedding and TF-IDF. Metric-based methods usually attempt to model metrics by capturing the temporal dependencies [50], [54], mining representative patterns [4], and learning inter-series relationships [6], [51]. [5] borrowed the spectral residual idea from visual saliency detection, making it easy to use in practice no matter whether labeled data exist. Recently, Chen *et al.* [4] achieved state-of-the-art by discovering anomalous metric patterns that sketch particular performance issues. Different from Hades, these methods only leverage single-source data, ignoring rich information from diverse sources of data and their interactions.

Some approaches employ multi-source data to identify software rollouts or operations that incur failures [41], [55]–[58]. [55]–[57] explored the correlations between logs and metrics to identify bad rolling upgrade operations. They regard logs as operation records and metric variations as the consequences of operations, instead of detecting anomalies via neck-and-neck fused information. [41], [58] leveraged logs and key performance indicators to alarm bad software changes, whereas [58] used each source separately rather than analyzing all sources generically, and [41] transformed logs into event occurrence series, and modeled series together with homogeneous indicators. However, these identifiers either regard metrics as a posteriori information rather than combing logs and metrics at the same level, or convert heterogeneous data into a homogeneous form ignoring the gap and high-order interactions among different data types. Hades outperforms these works by learning meaningful cross-modal heterogeneous representations while narrowing the log-metric gap.

IX. Conclusion

We study the manifestations of typical system anomalies on heterogeneous monitoring data and point out for the first time that logs and metrics have collaborative and complementary relationships in reflecting system anomalies, but neither of them only is sufficient. Motivated by the findings, we propose the first end-to-end semi-supervised approach, Hades, to detect anomalies effectively by exploiting heterogeneous information. Hades adopts semi-supervised learning to incorporate human oversight while reducing the annotation cost. It leverages multi-level log semantics and aspect-aware dependencies of metrics, all the while learning meaningful log-metric interactions via cross-modal attention. We evaluate Hades with comprehensive experiments on three datasets, and the results demonstrate that Hades outperforms all competitive approaches. Furthermore, the data, code, and experiment results in this study are released for replication.

Acknowledgement

The work described in this paper was supported by the National Natural Science Foundation of China (No. 62202511), and the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14206921 of the General Research Fund).

REFERENCES

[1] T. Maynard, "Cloud down impacts on the us economy," Lloyd's and AIR Worldwide, London, Technical Report, 2018. [Online]. Available: https://assets.lloyds.com/assets/pdf-air-cyber-lloyds-public-2018-final/1/pdf-air-cyber-lloyds-public-2018-final.pdf

[2] Z. Chen, Y. Kang, L. Li, X. Zhang, H. Zhang, H. Xu, Y. Zhou, L. Yang, J. Sun, Z. Xu, Y. Dang, F. Gao, P. Zhao, B. Qiao, Q. Lin, D. Zhang, and M. R. Lyu, "Towards intelligent incident management: why we need it and how we make it," in *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*. ACM, 2020, pp. 1487–1497.

[3] J. Lou, Q. Lin, R. Ding, Q. Fu, D. Zhang, and T. Xie, "Software analytics for incident management of online services: An experience report," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*. IEEE, 2013, pp. 475–485.

[4] Z. Chen, J. Liu, Y. Su, H. Zhang, X. Ling, Y. Yang, and M. R. Lyu, "Adaptive performance anomaly detection for online service systems via pattern sketching," *CoRR*, vol. abs/2201.02944, 2022. [Online]. Available: https://arxiv.org/abs/2201.02944

[5] H. Ren, B. Xu, Y. Wang, C. Yi, C. Huang, X. Kou, T. Xing, M. Yang, J. Tong, and Q. Zhang, "Time-series anomaly detection service at microsoft," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*. ACM, 2019, pp. 3009–3017.

[6] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*. ACM, 2019, pp. 2828–2837.

[7] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J. Lou, M. Chintalapati, F. Shen, and D. Zhang, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*. ACM, 2019, pp. 807–817.

[8] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Plelog: Semi-supervised log-based anomaly detection via probabilistic label estimation," in *43rd IEEE/ACM International Conference on Software Engineering: Companion Proceedings, ICSE Companion 2021, Madrid, Spain, May 25-28, 2021*. IEEE, 2021, pp. 230–231.

[9] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. ACM, 2017, pp. 1285–1298.

[10] S. Nedelkoski, J. Bogatinovski, A. K. Mandapati, S. Becker, J. Cardoso, and O. Kao, "Multi-source distributed system data for ai-powered analytics," in *Service-Oriented and Cloud Computing*. Cham: Springer International Publishing, 2020, pp. 161–176.

[11] N. L. of Tsinghua University. (2021) 2021 international aiops challenge. [Online]. Available: http://iops.ai/competition_detail/?competition_id=17&flag=1

[12] F. Faghri, S. Bazarbayev, M. Overholt, R. Farivar, R. H. Campbell, and W. H. Sanders, "Failure scenario as a service (fsaas) for hadoop clusters," in *Proceedings of the Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management*, ser. SDMCMM '12. New York, NY, USA: Association for Computing Machinery, 2012. [Online]. Available: https://doi.org/10.1145/2405186.2405191

[13] S. He, J. Zhu, P. He, and M. R. Lyu, "Loghub: A large collection of system log datasets towards automated log analytics," *CoRR*, vol. abs/2008.06448, 2020. [Online]. Available: https://arxiv.org/abs/2008.06448

[14] B. Chen and Z. M. J. Jiang, "Characterizing and detecting anti-patterns in the logging code," in *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*. IEEE / ACM, 2017, pp. 71–81.

[15] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *arXiv preprint arXiv:1607.04606*, 2016.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 2017, pp. 5998–6008. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[17] C. Lea, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks: A unified approach to action segmentation," in *Computer Vision - ECCV 2016 Workshops - Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III*, ser. Lecture Notes in Computer Science, vol. 9915, 2016, pp. 47–54.

[18] B. Li, T. Yang, Z. Chen, Y. Su, Y. Yang, and M. R. Lyu. Hades. [Online]. Available: https://github.com/BEbillionaireUSD/Hades/

[19] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "Towards automated log parsing for large-scale log data analysis," *IEEE Trans. Dependable Secur. Comput.*, vol. 15, no. 6, pp. 931–944, 2018.

[20] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," *ACM Comput. Surv.*, vol. 54, no. 6, pp. 130:1–130:37, 2021.

[21] A. S. Foundation. (2018) Apache spark. [Online]. Available: https://spark.apache.org/

[22] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[23] I. Inc. (2004) Hibench. [Online]. Available: https://github.com/Intel-bigdata/HiBench

[24] M. Saadoon, S. h. Ab hamid, H. Sofian, H. Altarturi, N. Daud, Z. Azizul Hasan, A. Sani, and A. Asemi, "Experimental analysis in hadoop mapreduce: A closer look at fault detection and recovery techniques," *Sensors*, vol. 21, p. 3799, 05 2021.

[25] Yahoo. (2011) Anarchyape. [Online]. Available: https://github.com/david78k/anarchyape

[26] O. Denloy, Y. L. Lu, E. Kaczmarek, and A. Gruza. (2015) Pat. [Online]. Available: https://github.com/asonje/PAT

[27] C. I. King. (2020) Stress-ng. [Online]. Available: https://github.com/ColinIanKing/stress-ng

[28] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement*, vol. 20, pp. 37–46, 1960. [Online]. Available: https://w3.ric.edu/faculty/organic/coge/cohen1960.pdf

[29] P. He, Z. Chen, S. He, and M. R. Lyu, "Characterizing the natural language descriptions in software logging statements," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 178–189.

[30] D. Yuan, J. Zheng, S. Park, Y. Zhou, and S. Savage, "Improving software diagnosability via log enhancement," *ACM Transactions on Computer Systems (TOCS)*, vol. 30, no. 1, pp. 1–28, 2012.

[31] D. Yuan, S. Park, P. Huang, Y. Liu, M. M. Lee, X. Tang, Y. Zhou, and S. Savage, "Be conservative: Enhancing failure diagnosis with proactive logging," in *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, 2012, pp. 293–306.

[32] P. Huang, C. Guo, L. Zhou, J. R. Lorch, Y. Dang, M. Chintalapati, and R. Yao, "Gray failure: The achilles' heel of cloud-scale systems," in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, 2017, pp. 150–155.

[33] Z. Chen, J. Liu, W. Gu, Y. Su, and M. R. Lyu, "Experience report: Deep learning-based system log analysis for anomaly detection," *CoRR*, vol. abs/2107.05908, 2021. [Online]. Available: https://arxiv.org/abs/2107.05908

[34] Y. Huo, Y. Su, B. Li, and M. R. Lyu, "Semparser: A semantic parser for log analysis," *CoRR*, vol. abs/2112.12636, 2021. [Online]. Available: https://arxiv.org/abs/2112.12636

[35] X. Yu, P. Joshi, J. Xu, G. Jin, H. Zhang, and G. Jiang, "Cloudseer: Workflow monitoring of cloud infrastructures via interleaved logs," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2016, Atlanta, GA, USA, April 2-6, 2016*. ACM, 2016, pp. 489–502.

[36] X. Ding, Y. Li, A. Belatreche, and L. P. Maguire, "An experimental evaluation of novelty detection methods," *Neurocomputing*, vol. 135, pp. 313–327, 2014.

[37] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE International Conference on Web Services, ICWS 2017, Honolulu, HI, USA, June 25-30, 2017*. IEEE, 2017, pp. 33–40.

[38] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2019, Montreal, QC, Canada, May 25-31, 2019*. IEEE / ACM, 2019, pp. 121–130.

[39] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *CoRR*, vol. abs/1803.01271, 2018. [Online]. Available: http://arxiv.org/abs/1803.01271

[40] A. F. Agarap, "Deep learning using rectified linear units (relu)," *CoRR*, vol. abs/1803.08375, 2018. [Online]. Available: http://arxiv.org/abs/1803.08375

[41] N. Zhao, J. Chen, Z. Yu, H. Wang, J. Li, B. Qiu, H. Xu, W. Zhang, K. Sui, and D. Pei, "Identifying bad software changes via multimodal anomaly detection for online service systems," in *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*. ACM, 2021, pp. 527–539.

[42] I. Graf, U. Kressel, and J. Franke, "Polynominal classifiers and support vector machines," in *Artificial Neural Networks - ICANN '97, 7th International Conference, Lausanne, Switzerland, October 8-10, 1997, Proceedings*, ser. Lecture Notes in Computer Science, vol. 1327. Springer, 1997, pp. 397–402.

[43] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50.

[44] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[45] J. F. Bowring, J. M. Rehg, and M. J. Harrold, "Active learning for automatic classification of software behavior," in *Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 195–205.

[46] V. Le and H. Zhang, "Log-based anomaly detection without log parsing," *CoRR*, vol. abs/2108.01955, 2021. [Online]. Available: https://arxiv.org/abs/2108.01955

[47] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, and R. Zhou, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. ijcai.org, 2019, pp. 4739–4745.

[48] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults," in *31st IEEE International Symposium on Software Reliability Engineering, ISSRE 2020, Coimbra, Portugal, October 12-15, 2020*. IEEE, 2020, pp. 92–103.

[49] W. Meng, Y. Liu, Y. Huang, S. Zhang, F. Zaiter, B. Chen, and D. Pei, "A semantic-aware representation framework for online log analysis," in *29th International Conference on Computer Communications and Networks, ICCCN 2020, Honolulu, HI, USA, August 3-6, 2020*. IEEE, 2020, pp. 1–7.

[50] N. Laptev, S. Amizadeh, and I. Flint, "Generic and scalable framework for automated time-series anomaly detection," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*. ACM, 2015, pp. 1939–1947.

[51] Z. Li, Y. Zhao, J. Han, Y. Su, R. Jiao, X. Wen, and D. Pei, "Multivariate time series anomaly detection and interpretation using hierarchical inter-metric and temporal embedding," in *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*. ACM, 2021, pp. 3220–3230.

[52] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, "USAD: unsupervised anomaly detection on multivariate time series," in *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*. ACM, 2020, pp. 3395–3404.

[53] S. Tuli, G. Casale, and N. R. Jennings, "Tranad: Deep transformer networks for anomaly detection in multivariate time series data," *Proc. VLDB Endow.*, vol. 15, no. 6, p. 1201–1214, feb 2022.

[54] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Söderström, "Detecting spacecraft anomalies using lstms and non-parametric dynamic thresholding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*. ACM, 2018, pp. 387–395.

[55] M. Farshchi, J. Schneider, I. Weber, and J. C. Grundy, "Experience report: Anomaly detection of cloud application operations using log and cloud metric correlation analysis," in *26th IEEE International Symposium on Software Reliability Engineering, ISSRE 2015, Gaithersbury, MD, USA, November 2-5, 2015*. IEEE Computer Society, 2015, pp. 24–34.

[56] C. Luo, J. Lou, Q. Lin, Q. Fu, R. Ding, D. Zhang, and Z. Wang, "Correlating events with time series for incident diagnosis," in *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*. ACM, 2014, pp. 1583–1592.

[57] S. He, Q. Lin, J. Lou, H. Zhang, M. R. Lyu, and D. Zhang, "Identifying impactful service system problems via log analysis," in *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*. ACM, 2018, pp. 60–70. [Online]. Available: https://doi.org/10.1145/3236024.3236083

[58] Z. Li, Q. Cheng, K. Hsieh, Y. Dang, P. Huang, P. Singh, X. Yang, Q. Lin, Y. Wu, S. Levy, and M. Chintalapati, "Gandalf: An intelligent, end-to-end analytics service for safe deployment in large-scale cloud infrastructure," in *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*. USENIX Association, 2020, pp. 389–402. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/li

APPENDIX

A. Workloads

We conduct 16 workloads on our established cluster based on HiBench [23], falling into categories:

- Graph-based: Graph-X-based PageRank and NWeight calculation (an iterative graph-parallel algorithm).
- SQL-related operations: Join, Aggregation, and Scan.
- Websearch: PageRank.
- Basic: Sort (text inputs generated by RandomTextWriter), WordCount, TeraSort, Sleep, and Repartition (benchmarking shuffle performance).
- Machine learning: Naive Bayesian Classification, Alternating Least Squares, Latent Dirichlet Allocation, Random Forest, and Singular Value Decomposition

The above workloads cover a large range of general application scenarios to enhance the respectiveness of the collected data.

B. Data

We collect log files of 37.64MB in total. After cleaning and parsing these raw log messages with Drain [37], we obtain 1,048,576 log events with 151 log templates, covering 95.87 hours. The anomaly ratio of log events is 0.055%.

The dataset contains 11 metrics with a length of 64,422. Each metric is sampled per second. The metrics include: CPU usage of User, System, IOwait, and Idle; I/O related metrics including rkB/s, wkB/s, and util; used memory and Commit memory; communication metrics via a network including rxkB/s and txkB/s. The anomaly ratio of metrics is 24.47%.